Jan Lichtermann

# Allerlei Nützliches für den TRS-80

TRS-80-Besitzer müssen manche Kleinigkeit bei ihrem Computer in Kauf nehmen, die auf die Dauer ärgerlich ist. In den meisten Fällen kann man mit wenig Aufwand Abhilfe schaffen. Die im folgenden angesprochenen Tips und Kniffe haben sich in der Praxis vielfältig bewährt.

## Die Dauer eines Tastendrucks wird festgestellt

Die INKEY-Funktion hat den Nachteil, daß nach einmaligem Tastendruck die Dauer der Betätigung nicht ermittelt werden kann. Möchte man mit einer Taste, z. B. in einem Grafikprogramm, eine Dauerfunktion erreichen, so kann man die Tastatur mit der PEEK-Funktion direkt abfragen und in einer Programmschleife einen Befehl so lange ausführen, wie die Taste betätigt wird. Die *Tabelle* gibt Aufschluß darüber, welches Argument zur Abfrage einer bestimmten Taste erforderlich ist. Wird die Taste betätigt, so erhält man den zugehörigen Wert als Resultat der Abfrage. Ist keine Taste betätigt, ergibt die PEEK-Abfrage den Wert 0. *Bild 1* verdeutlich das Prinzip d Tastaturabfrage durch PEEK.

S aus einem bestimmten Grund, etwa zur Erhöhung der Aufmerksamkeit, eine Stringvariable durch einen blinkenden Cursor eingelesen werden, so empfiehlt sich die Nachbildung des INPUT-Befehls durch die INKEY-Funktion. Durch die Wahl der Schleifenlänge kann man die Blinkfrequenz des Cursors beeinflussen. *Bild 2* zeigt, wie das Einlesen eines Strings mit der INKEY-Funktion bewerkstelligt wird.

## Ausgabe im technischen Format

Obwohl das TRS-80-Basic die komfortable PRINT-USING-Anweisung beinhaltet, ergeben sich bei der Ausgabe von Zahlen manchmal Schwierigkeiten. Von der Technik her schätzt man das Arbeiten mit den Kürzeln piko ($10^{-12}$), nano ($10^{-9}$), Mega ($10^{6}$), Tera ($10^{9}$), Giga ($10^{12}$). Von Kürzel zu Kürzel unterscheidet sich der Exponent um jeweils 3. Eine automatische Anpassung wäre deshalb in vielen Fällen wünschenswert. Das in *Bild 3* dargestellte Unterprogramm erledigt das. Übergeben wird die auszugebende Zahl Z, ebenfalls verwendet werden die Variablen ZZ und V. Letzteres kann eine Ganzzahlvariable sein, lediglich ZZ und Z müssen vom selben Typ sein, also entweder von einfacher oder doppelter Genauigkeit.

## Tastenprellen und Ladeprobleme

Vielen mag das Tastaturprellen beim TRS-80 schon Kopfzerbrechen bereitet haben. Sind die Tastenkontakte verschmutzt, so werden Buchstaben zum Teil doppelt geschrieben. Tandy empfiehlt dagegen das Reinigen der Tastaturkontakte oder die Verwendung eines kleinen Maschinenspracheprogramms KBFIX. Dieses Programm konnte ich von Tandy kostenlos erhalten, und es arbeitet zu meiner vollen Zufriedenheit. Ist das Programm geladen, wird das Tastaturprellen wirkungsvoll per Software eliminiert.

Ein anderes beinahe noch kritischeres Problem ist das Laden der Programme vom Kassettenrecorder. Dabei ist die Lautstärke am Recorder peinlich genau

```
10 A=PEEK(14337):IF A=2 THEN PRINT"A";:GOTO 10
20 IF A=4 THEN PRINT"B";:GOTO 10
30 IF A=8 THEN PRINT"C";:GOTO 10
40 GOTO10
```

**Bild 1. Das Programm gibt so lange einen der Buchstaben A, B oder C aus, wie die entsprechende Taste gedrückt ist**

**Tabelle zur Tastaturabfrage**

| Zeichen | PEEK() | Wert |
|---|---|---|
| @ | 14337 | 1 |
| A | 14337 | 2 |
| B | 14337 | 4 |
| C | 14337 | 8 |
| D | 14337 | 16 |
| E | 14337 | 32 |
| F | 14337 | 64 |
| G | 14337 | 128 |
| H | 14338 | 1 |
| I | 14338 | 2 |
| J | 14338 | 4 |
| K | 14338 | 8 |
| L | 14338 | 16 |
| M | 14338 | 32 |
| N | 14338 | 64 |
| O | 14338 | 128 |
| P | 14340 | 1 |
| Q | 14340 | 2 |
| R | 14340 | 4 |
| S | 14340 | 8 |
| T | 14340 | 16 |
| U | 14340 | 32 |
| V | 14340 | 64 |
| W | 14340 | 128 |
| X | 14344 | 1 |
| Y | 14344 | 2 |
| Z | 14344 | 4 |
| frei | 14344 | 8 |
| frei | 14344 | 16 |
| frei | 14344 | 32 |
| frei | 14344 | 64 |
| frei | 14344 | 128 |
| 0 | 14352 | 1 |
| 1 ! | 14352 | 2 |
| 2 " | 14352 | 4 |
| 3 # | 14352 | 8 |
| 4 $ | 14352 | 16 |
| 5 % | 14352 | 32 |
| 6 & | 14352 | 64 |
| 7 ' | 14352 | 128 |
| 8 ( | 14368 | 1 |
| 9 ) | 14368 | 2 |
| : * | 14368 | 4 |
| ; + | 14368 | 8 |
| , < | 14368 | 16 |
| − = | 14368 | 32 |
| . > | 14368 | 64 |
| / ? | 14368 | 128 |
| ENTER | 14400 | 1 |
| CLEAR | 14400 | 2 |
| BREAK | 14400 | 4 |
| ↑ | 14400 | 8 |
| ↓ | 14400 | 16 |
| ← | 14400 | 32 |
| → | 14400 | 64 |
| SPACE | 14400 | 128 |
| SHIFT | 14464 | 1 |
| frei | 14464 | 2 |
| frei | 14464 | 4 |
| frei | 14464 | 8 |
| frei | 14464 | 16 |
| frei | 14464 | 32 |
| frei | 14464 | 64 |
| frei | 14464 | 128 |

```
10 B$=""
20 PRINT CHR$(14);:FOR N=1TO50:A$=INKEY$:IF A$=""THEN NEXT ELSE GOSUB 100:NEXT
30 PRINT CHR$(15);:FOR N=1TO50:A$=INKEY$:IF A$=""THEN NEXT ELSE GOSUB 100:NEXT
40 GOTO 20
100 IF ASC(A$)=13 THEN PRINT CHR$(14);A$;:GOTO 130
110 IF ASC(A$)=8 AND LEN(B$)>0 THEN B$=LEFT$(B$,LEN(B$)-1):PRINTA$;:RETURN
120 IF ASC(A$)<>8 THEN B$=B$+A$:PRINTA$;
125 RETURN
130 REM STRING STEHT ALS B$ BEREIT, NACH BEARBEITUNG MIT RETURN ZURUECKKEHREN !
```

**Bild 2. Das Programm liest einen String ein, dabei wird ein blinkender Cursor erzeugt – Enter schließt den String ab**

```
60000 IF Z=0 THEN V=1:GOTO60010 ELSE ZZ=ABS(Z):V=FIX(LOG(ZZ)/LOG(10))-FIX(LOG
      (ZZ)/LOG(10)/3)*3+2:IF ZZ<1 THEN V=V+2
60010 PRINTUSING STRING$(V,"#")+".###↑↑↑↑";Z:RETURN
```

**Bild 3. Das Unterprogramm gibt die Zahl Z im technischen Format aus**

einzustellen, da sonst immer wieder Ladefehler auftreten. Auch ist die Lautstärke je nach Bandsorte verschieden zu wählen. Um so erfreuter war ich, als ich las, daß Tandy eine Zusatzplatine einbaut. Der kostenlose Einbau (er erforderte zwar einige Schreibarbeit meinerseits, da die Firma Tandy nicht sofort reagierte) hat sich durchaus gelohnt. Ladefehler treten jetzt nur noch in Ausnahmefällen auf.

Weitere Hardwareverbesserungen einfachster Art sind der zusätzliche Einbau zweier Tastschalter rechts neben der Tastatur. In der Klappe zur Erweiterung des TRS-80 befindet sich ein Taster, der beim Schieflaufen des Ladevorgangs bzw. eines Maschinenspracheprogramms betätigt werden kann und einen nichtmaskierbaren Interrupt am Prozessor auslöst. Die Betätigung dieses Tasters ist recht umständlich: Man muß jedesmal die Klappe öffnen und ihn auf der Rückseite mühsam ertasten. Deshalb kann man einen einfachen Taster parallel dazu legen und neben der Tastatur montieren. Dies empfiehlt sich sehr für Experimente mit Maschinensprache. Der Prozessor kann damit jederzeit ohne Programmverlust unterbrochen werden.

Bekannt ist jedem TRS-80-Anwender die nach dem Einschalten des TRS-80 erscheinende Frage: MEMORY SIZE?, die zur Reservierung eines vor Basic geschützten Arbeitsbereichs für Maschinensprache dient. Ausgelöst wird dieser Vorgang hardwaremäßig nach dem Einschalten durch den sogenannten Power-On-Reset. Arbeitet man viel mit Maschinensprache, so sind manchmal Umreservierungen nötig. Dies würde ein Aus- und Einschalten der Anlage erforderlich machen, wobei die Anlage einige Sekunden ausgeschaltet bleiben muß. Hier kann man einen weiteren Taster einbauen, der diesen Power-On-Reset simuliert. Dadurch braucht man die Anlage nicht an der Rückseite ein- und auszuschalten.

Betrachten wir einmal den Schaltungsausschnitt in *Bild 4*: Wird das Gerät eingeschaltet, so ist C42 entladen; am Eingang von Z53 liegt also L-Pegel an, damit auch am Ausgang von Z52. Dies veranlaßt den Prozessor dazu, mit der Befehlsausführung bei Adresse 0000 zu beginnen, sobald der $\overline{\text{Reset}}$-Eingang wieder H-Pegel erhält. Dies geschieht dadurch, daß C42 über R47 langsam aufgeladen wird. Im Betriebszustand ist C42 also geladen. Legt man jetzt einen Taster parallel zu C42, kann man ihn auch während des Betriebs entladen und damit den Einschaltvorgang simulieren. Dazu muß man C42 auf der Platine suchen und eine entsprechende Leitung anlöten. Ich verwende hierzu eine abgeschirmte Leitung, damit im Videoteil keine wilden Schwingungen ausgelöst werden. Doch Achtung! Versehentliches Betätigen dieses Tasters löscht ein Basic-Programm im Speicher. Auch verliert man durch einen solchen Eingriff die Garantie des Herstellers.

## Der Speicher wird erweitert

Bekanntlich kann ein 4-KByte-Modell auf 16 K erweitert werden. Dazu werden die Speicherchips ausgetauscht. Zusätzlich müssen zwei Shunts ausgetauscht werden. Führt man diese Arbeiten selbst aus, kann man eine Menge Geld sparen! Benötigt werden acht Chips vom Typ 4116 (450 ns oder schneller). Es handelt sich um dynamische NMOS-RAMs. Beim Einbau sind unbedingt die Handhabungsregeln von MOS-Bausteinen zu beachten, d. h. auf jeden Fall ist statische Aufladung zu vermeiden. Man entfernt die alten Speicherchips Z13 bis Z20 und setzt unter Beachtung der Richtung (Kerbe) die neuen ICs ein. Dann wechselt man die DIP-Shunts A3 und A71 aus. Das folgende gilt nur für die D-Version des TRS-80 (letzter Buchstabe der Nr. an Unterseite des Gehäuses) und „BASIC LEVEL II": Bei A3 sind sämtliche Kontakte zu überbrücken, bei A71 sind Brücken gemäß *Bild 5* herzustellen. Möchte man keine Shunts kaufen, so kann man die Verbindung mit Schaltdraht herstellen. Korrekte Verbindungen sind die erste Voraussetzung für den Betrieb, sonst läuft der Computer nicht mehr. Hat man eine andere Platinenversion oder kein Level-2-Basic, so sind die Brücken anders herzustellen! Die Funktion des eingebauten neuen Speichers läßt sich durch die Eingabe von PRINT MEM grob überprüfen: Bei leerem Arbeitsspeicher muß die Zahl 15572 erscheinen.

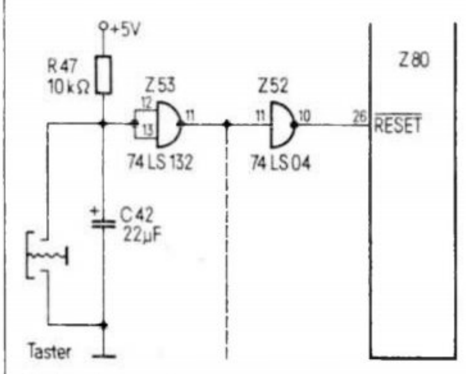



**Bild 4. So wird der Power-On-Reset beim TRS-80 erzeugt**

### Literatur


[1] TRS-80 micro computer technical reference handbook. Radio Shack.

[2] Stübs, *Martin*: Programmieren mit TRS-80. Ing. W. Hofacker GmbH.

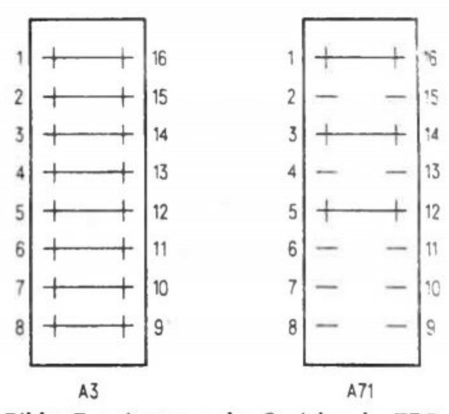




**Bild 5. Erweitert man den Speicher des TRS-80 von 4 KByte auf 16 KByte, dann müssen bei der D-Version (letzter Buchstabe der Nummer auf der Unterseite des Gehäuses) die eingezeichneten Verbindungen hergestellt werden**

lich, wenn auch in etwas abgewandelter Form. Andererseits weist das Microsoft-Basic ein paar unerklärliche Schwächen auf. Beispielsweise entdeckten wir im Zusammenhang mit der Rechengeschwindigkeit, daß bei Verwendung von Zahlen in Rechenoperationen nur ganzzahlig gerechnet wird (beispielsweise ist bei B = 10/3 das Resultat B = 3). Man muß also vorher die Zahlen in Variable stecken und die Operationen möglichst ausschließlich mit Variablen durchführen: also bitte nur B/2 oder so etwas!

Zum anderen ließ sich einfach kein Trick finden, wie File-Namen auf der Diskette per Variable definiert werden können, das Testobjekt verlangte zum Schreiben von Files grundsätzlich eine Direkteingabe per Tastatur oder Programm. So etwas war beim Atari-Basic hingegen problemlos möglich gewesen.

### Andere Programmiersprachen

Nicht verschwiegen werden soll, daß es auch noch ein Pascal und einen Assembler für die Atari-Mikros gibt sowie eine ganze Palette von Hardware-Zubehör. Ebenfalls zu erwähnen sind die verschiedenen Spiele (nicht zu verwechseln mit den Video-Spielen), die zum Teil die Grafikmöglichkeiten bis zum Optimum nutzen und auf dem Bildschirm Erstaunliches bewerkstelligen. Vielleicht doch hin und wieder eine Möglichkeit für den geplagten Familienvater und angehenden Computerspezialisten, mit Beistand des Juniors so ein Gerät in die Wohnstube zu bringen...

### Immer mit der Ruhe

Beim Umgang mit der Diskettenstation 810 kommt einem immer wieder dieser Satz in den Sinn. Als 100-m-Sprinter hätte sie jedenfalls keine Chance. Andererseits gab es auch keine Schwierigkeiten mit Schreib- oder Lesefehlern auf jeglichen Disketten oder bei der Datenübertragung. Dieses Konzept ist wohl mehr auf Betriebssicherheit ausgelegt. Setzt man das Laufwerk in Aktion, so wird man auch nie über die Operationsdauer im Unklaren gelassen, es ist nicht zu überhören.

Das DOS zum Betrieb der Diskettenstation ist recht umfangreich, es ist aber nicht immer vollständig im Arbeitsspeicher des Rechners. Im Rechner selbst befinden sich nur die Teile, die zum Schreiben und Lesen von Programmen oder Dateien erforderlich sind. Es gibt aber noch eine ganze Reihe von nützlichen Routinen zum Duplizieren von Dateien oder ganzen Disketten, umbenennen von Dateien oder Schützen derselben vor Schreibzugriff und einiges andere mehr. Braucht man diese Routinen, so werden diese hinzugeladen.

Doch was passiert, wenn man dies inmitten einer Programmentwicklung benötigt (man hat beispielsweise keine formatierte Diskette zum Speichern des Programmes mehr)? Der Inhalt des Rechnerspeichers wird dann auf die momentan einliegende Diskette gerettet. Auf dieser ist ein Bereich speziell für diesen Zweck reserviert. Ist die Diskette formatiert, wird das Programm wieder in den Rechner zurückgeholt (von der anderen Diskette natürlich). Auf der neuen Diskette läßt sich das Programm nun regulär ablegen. Man kann den Bereich zum Retten des Rechnerspeicher-Inhaltes auch weglassen und gewinnt damit zusätzlichen Platz auf der Diskette, muß aber in Kauf nehmen, unter Umständen auch einmal einen Speicherinhalt zu verlieren.

Aber nicht ganz ungefährlich ist dieses Verfahren: Durch Vertauschen der Disketten ist es möglich, einen leeren Speicher auf die Diskette zu übertragen und eventuell vorhandene Daten zu überschreiben. Da die Diskettenstation alle Disketten im richtigen Format akzeptiert, d. h. mit der richtigen Formatierung, wird dieser Fehler auch nicht erkannt. Bei derartigen Aktionen ist also immer höchste Vorsicht geboten, denn leicht ist die Arbeit von Stunden zunichte gemacht.

### Grafik ist Trumpf

Ganz sicher ist ein Mikrocomputer wie der Atari-800 nicht jedermanns Geschmack. Wie immer muß man abwägen, welche Eigenschaften für den Anwender am wichtigsten sind. Als Spielgerät allein ist er sicher zu teuer, aber in der grafischen Darstellung ist er einer der Besten in seiner Klasse. Jeder spricht heute davon, daß die Grafik in Zukunft mehr und mehr Bedeutung haben wird. Nun, hier ist einer, der es bereits kann.

### Literatur

[1] *Andree, Hans-Joachim:* Kurvenentwicklung auf Bildschirm und Plotter. mc 1982, Heft 5, Seite 42.

# Repeat-Funktion für TRS-80

Die Repeat-Funktion, d. h. die wiederholte Ausgabe eines Zeichens bei längerem Tastendruck, ist zwar eine sehr nützliche Angelegenheit, aber leider bei den meisten Tischcomputern nicht vorhanden. Sie kann durch Software aber leicht nachgerüstet werden.

Ein Maschinenprogramm, das diese Funktion beim TRS-80 ermöglicht, wird vom abgedruckten Basic-Programm erzeugt. Nachdem es geladen und initialisiert ist, wird der Tastendruck „wiederholt", sobald eine Taste länger als 0,5 Sekunden gedrückt ist. Die Wiederholfrequenz beträgt ca. 15 Hz. Die Zeit bis zum Einsatz der Wiederholung und die Wiederholfrequenz können mit POKE 32764,X bzw. mit POKE 32738,X geändert werden.

Besonders beim Editieren hat sich die Repeat-Funktion als sehr nützlich erwiesen, da mit ihrer Hilfe der Cursor schnell und einfach an die richtige Stelle gebracht werden kann.
Bevor das Programm geladen und gestartet wird, muß genügend Speicherplatz durch MEM SIZE 32694 reserviert werden.

*Luidger Röckrath*

```
10 REM COPYRIGHT BY LUIDGER ROECKRATH,
20 REM REPEATFUNKTION
30 REM MEM SIZE: 32694
40 DATA33,196,127,34,22,64,217,22,0,217,195,204,6,33,54,64,1,1
50 DATA56,22,0,10,95,174,115,32,20,20,44,203,1,242,204,127,217
60 DATA43,124,181,62,0,32,3,46,128,122,217,201,95,197,1,0,5,205
70 DATA96,0,193,10,163,40,6,122,7,7,205,254,3,217,33,0,4,87,217
80 DATA201
90 FORI=32695TO32767:READA:POKEI,A:NEXT
100 POKE16526,183:POKE16527,127:A=USR(0)
```

**Dieses Basic-Programm erzeugt den Maschinencode für die Repeat-Funktion**

Reinhard Grabowski

# Rekursive Programmierung in Basic

**Rekursion – was ist das? Den meisten dürfte schon einmal ein Fernsehbild begegnet sein, das einen Fernsehbildschirm zeigt, auf dem wiederum der gleiche Bildschirm abgebildet ist, der wiederum den gleichen Bildschirm enthält. Ein solches – sich selbst abbildendes – Bild ist ein rekursives Bild und dient als anschauliches Muster für einen sich selbst wiederholenden Vorgang.**

Auch bei der Programmierung sind sich selbst wiederholende Rechenprozesse oder, allgemeiner, Datenverarbeitungsprozesse denkbar. Und nicht nur denkbar: Sie werden auch in der Praxis angewandt, wenn auch – nach den Erfahrungen des Verfassers – bislang nur in bescheidenem Umfang. Ein einfacher, vertrauter rekursiver Rechenvorgang ist eine sich selbst aufrufende Funktion. Zum Beispiel in der Anweisung

LET Y = EXP(EXP(EXP(X))).

Alle besseren Basic-Interpreter „verstehen" diese Anweisung. (Sie kann aber bei größerem Argument X zu einem Überlauf führen!)

## GOSUB und Rekursion

Diese Art des rekursiven Aufrufes einer Funktion wird als derart selbstverständlich angesehen, daß er meist nicht mit dem Begriff Rekursion verknüpft wird. Weniger selbstverständlich ist der Vorschlag, in einem (durch GOSUB aufgerufenen) Unterprogramm mit GOSUB dieses Programm erneut aufzurufen, also ein Unterprogramm der Gestalt

```
1000 REM Beginn Unterprogramm
1001 ...

....
1100 GOSUB 1000
1999 RETURN: REM Ende Unterpro-
        gramm
```

zu schreiben.

Sie als Leser werden vielleicht spontan einwenden, daß diese Art der Programmierung in Basic nicht möglich sei. Auf meinen Hinweis, daß für die Ausführung der Anweisung

1100 GOSUB 1000

als Sprungbefehl mit Rücksprung kein formaler Hinderungsgrund vorliege, werden Sie nach einigem Überlegen vielleicht einwenden, daß dieser Sprungbefehl mit jedem erneuten Aufruf einen weiteren Aufruf einschließt und daher das Aufrufen kein Ende findet. Allerdings wird ein nicht programmgemäßes Ende erzwungen durch eine Fehlermeldung mit dem Hinweis, daß der für die GOSUB-Verwaltung zur Verfügung stehende Speicherbereich voll besetzt sei. Dieser Einwand ist berechtigt. In der Tat, wenn eine Aufgabe in rekursiver Programmierung gelöst werden kann, dann kann die Aufgabe nur dann sinnvoll gelöst werden, wenn innerhalb des Programmes eine Entscheidung vorgesehen ist, daß nach Erfüllung einer Bedingung das rekursive Aufrufen ein Ende finde. Die Erfüllung einer solchen Bedingung kann zum Beispiel durch den Wert „wahr" (entsprechend −1 in der logischen Variablen LV) gekennzeichnet werden. Dann bewirkt zum Beispiel eine Anweisung der Form

1099 IF LV < > 0 THEN GOTO 1999,

daß die Folge rekursiver Aufrufe unterbrochen wird.

## Zurück mit RETURN

Vielleicht haben Sie als Leser aber gedankliche Schwierigkeiten einzusehen, daß nach den mehrfach wiederholten (Vorwärts-)Sprüngen bei GOSUB die entsprechende Anzahl von Rücksprüngen durchgeführt wird, weil diesen geschachtelten (Vorwärts-)Sprüngen keine geschachtelten Anweisungen RETURN entgegenstehen. Bei näherer Betrachtung wird jedoch erkennbar, daß durchaus eine einzige Anweisung RETURN genügt.

Dies sei am Schema in *Tabelle 1* erläutert. Beim erstmaligen Aufruf des Unterprogramms (durch ein übergeordnetes Programm) wird das Unterprogramm zunächst bis zur Anweisung 1100 „abgearbeitet", dargestellt durch einen Pfeil parallel zur Abfolge der Anweisungs-Nummern. Dann erfolgt ein erster Selbstaufruf, also ein (Rück-)Sprung zur Anweisung 1000. Dies kennzeichnen wir im Schema durch einen schrägen Pfeil hin zur ersten Zeile in einer benachbarten „Ebene". Wiederum wird das Programm bis zur Anweisung 1100 abgearbeitet, wiederum durch einen Pfeil parallel zur Anweisungsfolge gekennzeichnet. Dieser Vorgang wiederholt sich so oft, bis

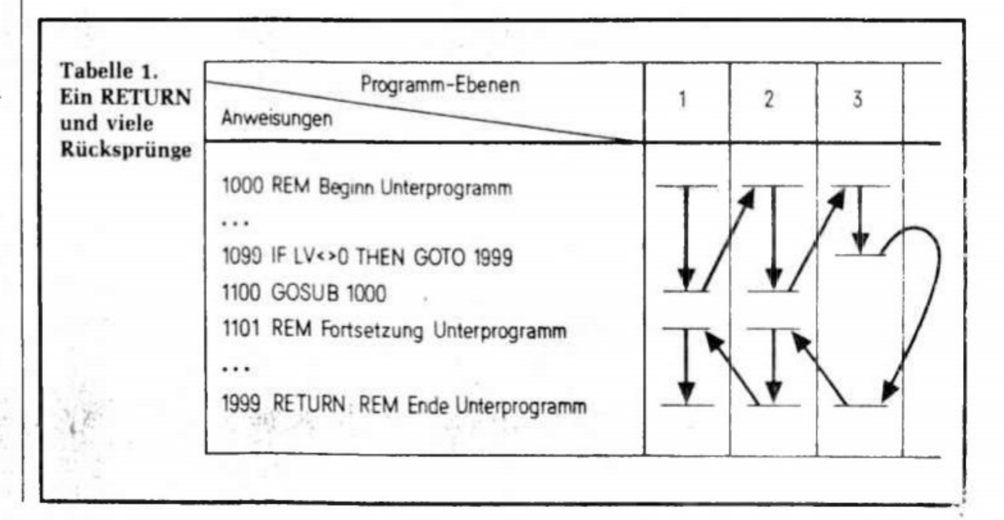



**Tabelle 1. Ein RETURN und viele Rücksprünge**

schließlich in einer Ebene, in unserem Schema in der Ebene 3, die Bedingung LV = −1 (wahr) erfüllt ist. Die bedingte Sprunganweisung 1099 bewirkt eine weitere Bearbeitung ab der Anweisung 1999 RETURN, die einen Rücksprung zu der Anweisung nach dem letztmals vorgefundenen GOSUB bewirkt, also zur Anweisung 1101 in Ebene 2. Der darauf folgende Programmteil wird abgearbeitet; die letzte Anweisung, 1999 RETURN, bewirkt wiederum einen Rücksprung zu der Stelle des von dieser Ebene gesehenen letzten Aufrufes. Die entsprechenden Pfeile in unserem Schema zeigen, daß durch die Rücksprünge schließlich wieder die Programm-Ebene 1 und danach wieder das übergeordnete Programm erreicht wird.

### Der Rückkehr-Adreß-Keller ist begrenzt

Entscheidend für den korrekten Ablauf der Sprünge und Rücksprünge ist offensichtlich die Eigenschaft des Basic-Interpreters, sich alle Aufrufe mit GOSUB zu merken und auch deren Reihenfolge, um beim Auftreffen auf eine RETURN-Anweisung zu demjenigen GOSUB zurückzuspringen, das in der Merkliste als das letzte vermerkt ist. Danach wird dieses GOSUB in der Merkliste gelöscht, so daß ein vorhergehendes GOSUB in der Liste nun als das letzte erscheint. Die Merkliste, also der zugehörige Speicherbereich, ist bei den Basic-Interpretern beschränkt und nicht sehr groß. Die maximale Anzahl, die vermerkt werden kann, ist von Rechnertyp zu Rechnertyp verschieden. Typisch ist eine Anzahl von 20.
Es scheint nun, daß der rekursive Aufruf von Unterprogrammen in Basic kein Problem darstellt. Das ist allerdings nur bedingt richtig. Denn wir müssen noch beachten, daß Basic keine echten Unterprogramme ermöglicht, sondern nur Sprünge mit Rücksprüngen. Bei echter Unterprogrammtechnik werden die innerhalb des Unterprogramms aufgetretenen Variablen völlig unabhängig von gleichnamigen Variablen im übergeordneten Programm und völlig unabhängig von anderen Unterprogrammen behandelt. Man sagt auch, daß sie nur lokal definiert seien. Der Inhalt einer gleichnamigen Variablen im aufrufenden Programm wird beim Aufruf des Unterprogrammes vom Betriebssystem sozusagen verwahrt, solange, bis das Unterprogramm abgearbeitet ist. Während des Unterprogramm-Laufes nimmt dann diese Variable Inhalte an, die vom Ablauf des Unterprogrammes vorgesehen sind. Bei Rückkehr in das aufrufende Programm wird automatisch der zwischenzeitlich verwahrte Wert dieser Variablen wieder neu zugewiesen.

### Variablen: In Basic immer global

Dieses zwischenzeitliche Verwahren kennt Basic nicht, eine lokale Definition von Variablen ist in Basic nicht möglich. Variablen gleichen Namens werden im aufrufenden Programm und im Unterprogramm nicht getrennt behandelt. Daß in Basic die Variablen nicht lokal, sondern immer nur für das gesamte Programm – also global – definiert werden können, ist ein Umstand, den wohl jeder Basic-Programmierer bei der Fehlersuche hat erfahren und sich einprägen müssen. Bei rekursivem Aufruf ist dieser Umstand besonders kritisch, denn zwangsläufig sind hier alle Variablennamen in allen Aufruf-Ebenen gleich. Der Programmierer hat, wenn er die rekursive Programmiertechnik nicht scheut, selbst durch geeignete zusätzliche Anweisungen dafür zu sorgen, daß die im zweiten Programmteil einer Ebene noch benötigten Variablen-Inhalte aus dem ersten Programmteil dieser Ebene wieder verfügbar sind. Der Programmierer muß also selbst die Verwahrung übernehmen. Dies ist mit Feldvariablen möglich, deren jeweiliger Index die Programm-Ebene kennzeichnet, aus welcher der Wert stammt, der an dieser Stelle des Feldes gespeichert, also bis zur Rückkehr in dieser Ebene verwahrt wird. Sinnvoll bleibt das natürlich nur, wenn nicht zuviele Werte je Ebene verwahrt werden müssen.

### Ein Beispiel zur Rekursion

Ein instruktives Beispiel für eine rekursive Anwendung eines Algorithmus, also einer Rechenvorschrift oder einer Datenverarbeitungsvorschrift, bietet die Lösung des folgenden Problems:
Eine Zahlenfolge Z(I), I = 0...N−1, bestehe aus $N = 2^M$ Elementen. Sie soll so umgeordnet werden, daß diejenigen Elemente, die auf Plätzen mit geradem Index liegen, auf die obere Hälfte der Plätze, also auf die Plätze mit den Indizes I = 0...(N/2)−1 zu liegen kommen. Notwendig müssen dann die Elemente, welche zu Anfang auf den Plätzen mit ungeradem Index lagen, in die untere Hälfte gerutscht sein. Dabei soll aber die Ordnung in beiden Teilfolgen erhalten bleiben.
Nehmen wir als Zahlenfolge die Zahlen von 0 bis 7, also mit $2^3 = 8$ Elementen. Anfangsfolge und resultierende Folge sind in *Tabelle 2* dargestellt.
Ein derartiges Umordnen ist auf vielerlei Art möglich. Wir verlangen aber, daß für das Umordnen kein nennenswerter zusätzlicher Speicherplatz benötigt wird, daß also die Umordnung durch paarweises Vertauschen vollzogen wird, und wir verlangen weiter, daß ein möglichst schnelles Verfahren gefunden wird.
Diese Aufgabenstellung tritt bei der numerischen Fouriertransformation reellwertiger diskreter Signale auf.
Ein effizienter Algorithmus mit paarweiser Vertauschung soll im folgenden dargelegt werden. Dazu werden die umzuordnenden Zahlen in *Tabelle 3* binär dargestellt.
Man erkennt, daß die neue Folge sich von der alten dadurch unterscheidet, daß die erste und die letzte Spalte in der

**Tabelle 2**

| Platzindex | Zahlenfolge vorher | Zahlenfolge nachher |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 1 | 2 |
| 3 | 2 | 4 |
| 4 | 3 | 6 |
| 5 | 4 | 1 |
| 6 | 5 | 3 |
| 7 | 6 | 5 |
| 8 | 7 | 7 |

**Tabelle 3**

| Platzindex | Zahlenfolge vorher | Zahlenfolge nachher |
|---|---|---|
| 1 | 000 | 000 |
| 2 | 001 | 010 |
| 3 | 010 | 100 |
| 4 | 011 | 110 |
| 5 | 100 | 001 |
| 6 | 101 | 011 |
| 7 | 110 | 101 |
| 8 | 111 | 111 |

**Tabelle 4**

| Platzindex | | | |
|---|---|---|---|
| 1 | 000 | 000 | 000 |
| 2 | 001 | 100 | 010 |
| 3 | 010 | 010 | 100 |
| 4 | 011 | 110 | 110 |
| 5 | 100 | 001 | 001 |
| 6 | 101 | 101 | 011 |
| 7 | 110 | 011 | 101 |
| 8 | 111 | 111 | 111 |

Binär-Darstellung vertauscht sind. Wie läßt sich dieses Resultat durch paarweises Vertauschen der Zahlen erreichen? Naheliegend ist der Gedanke, die Zahlen so zu vertauschen, daß je eine ungerade Zahl (Endziffer 1) in der oberen Hälfte mit einer geraden Zahl (Endziffer 0) in der unteren Hälfte vertauscht wird. In *Tabelle 4* ist das Ergebnis dieser Operationsfolge in der zweiten Spalte zu sehen. Dieser Vertauschungsvorgang läßt sich wie folgt benennen:

„Zweites Element aus erster Hälfte mit erstem Element aus zweiter Hälfte vertauschen; danach jeweils übernächstes Element aus erster Hälfte mit übernächstem Element aus zweiter Hälfte vertauschen".

Mit diesem Vertauschungsvorgang ist aber unser Endresultat noch nicht erreicht. Wir erreichen es, wenn der genannte Vertauschungsvorgang wiederholt wird, allerdings nicht für die gesamte Folge, sondern getrennt für die obere und die untere Teilfolge. Dann resultiert die Folge in der dritten Spalte der Tabelle 4, die ersichtlich das Resultat darstellt, wie ein Vergleich mit Tabelle 3 zeigt. Man überzeuge sich, daß der genannte Vertauschungsvorgang auch bei $2^4 = 16$ Elementen zum Ziel führt. In diesem Fall ist ein weiterer mit einer Halbierung verbundener Durchgang erforderlich.

### Der Algorithmus...

Der mit Worten beschriebene Vertauschungsvorgang ist der hier anwendbare Algorithmus, der wiederholt angewendet wird, und zwar zuerst auf die ganze Folge und danach auf die durch Halbierung entstehenden Teilfolgen. Die Aufeinanderfolge der Operationen

- Algorithmus auf alle Teilfolgen anwenden;
- alle Teilfolgen in je zwei weitere Teilfolgen zerlegen,

ersichtlich eine geschachtelte Operationsfolge, läßt sich einfach programmieren, indem man sie rekursiv programmiert. Die Vertauschung eines Paares wird mit folgender Verbundanweisung erreicht:

```
440   REM     Vertauschung
441   ZS      = Z(Z1) :
      Z(Z1)   = Z(Z2) :
      Z(Z2)   = ZS
```

Die Variablen Z1 und Z2 enthalten das Indexpaar, zwischen denen vertauscht werden soll, ZS ist eine Hilfsvariable zum Zwischenspeichern. Dieser Vorgang muß natürlich für alle relevanten Indizes in der oberen und unteren Hälfte der aktuellen Teilfolge wiederholt werden. Die Wiederholung läßt sich durch die Anweisung FOR...NEXT programmieren:

```
400   REM     Vertauschungsfolge
410   FOR ZI  = 0 TO ZL/4 - 1
420   Z1      = ZI + ZI + 1 + ZA
430   Z2      = Z1 - 1 + ZL/2
440   REM     Vertauschung
      (siehe oben)
450   NEXT
```

Die Variable ZI ist der Schleifenindex, in ZL ist die Länge der aktuell betrachteten Teilfolge gespeichert, also die Anzahl der Elemente in der Teilfolge, die – weil aus einer Teilfolge mit $2^{ZM}$ Elementen durch Halbieren hervorgehend – wiederum eine Potenz zur Basis 2 ist. ZA enthält den Anfangsindex der aktuellen Teilfolge. Die Anweisungen zur Berechnung des Indexpaares Z1 und Z2 zeigen, daß Z1 immer eine gerade Zahl und Z2 immer eine ungerade Zahl ist, wenn ZA gerade ist. Letzteres ist aber immer der Fall, weil ZA der Anfangsindex einer durch Halbieren entstandenen Teilfolge ist. Der Programmteil

```
400 REM Vertauschungsfolge
....
```

repräsentiert unseren Algorithmus, angewandt auf eine Teilfolge mit Anfangsindex ZA.

### ... ist rekursiv

Die Halbierung der Anfangsfolge bzw. einer Teilfolge in weitere zwei Teilfolgen und die Anwendung des Algorithmus führen wir geschachtelt durch. Damit ist gemeint, daß die Anfangsfolge – auf die unser Algorithmus angewandt worden ist – in zwei Teilfolgen zerlegt wird und zunächst nur die obere Teilfolge dem Algorithmus unterworfen und weiter in zwei Teilfolgen zerlegt wird. Dabei wird wiederum nur die jeweils obere der neu entstandenen Teilfolgen der genannten Prozedur unterworfen, und zwar solange, bis beim Teilungsvorgang nur eine Teilfolge mit zwei Elementen resultieren würde, für die unser Algorithmus nicht mehr relevant ist.

```
7600 REM UMORDNEN
7601 REM EINGABE:ZM,Z(I) I=[0,2^ZM-1]  AUSGABE:Z(I)
7602 REM INTERNE VARIABLE:ZA,ZI,ZJ,ZK,ZL,Z1,Z2,ZA(J,K),ZL(J,K)
7603 REM UNTERPROGRAMM IN REKURSIVER PROGRAMMIERUNG ZUM UMORDNEN EINER ZAHLEN
7604 REM FOLGE Z(I) MIT EINER ANZAHL VON 2^ZM ELEMENTEN WIE FOLGT:
7605 REM DIE TEILFOLGE MIT GERADEN INDIZES, Z(2*J), WIRD AUF DIE ERSTE HAELFTE
7606 REM DER PLAETZE GELEGT, DIE RESTLICHE TEILFOLGE, Z(2*J+1), AUF DIE ZWEITE
7607 REM HAELFTE. DIE ORDNUNG INNERHALB DER TEILFOLGEN BLEIBT ERHALTEN.
7608 REM DENKT MAN SICH DIE INDIZES BINAER GEGEBEN, IST DIESE UMORDNUNG EINE
7609 REM ZUSAMMENFASSUNG DER ELEMENTE MIT NIEDERWERTIGSTEM BIT 0 IN DER ERSTEN
7610 REM HAELFTE, DER ELEMENTE MIT NIEDERWERTIGSTEM BIT 1 IN DER ZWEITEN.
7612 REM DIE ANZAHL DER REKURSIONSEBENEN IST DURCH IMPLIZITE DIMENSIONIERUNG
7613 REM AUF 11 BESCHRAENKT! PROGRAMM GGF. MIT DIM ZA(N,M).ZL(N,M) ERWEITERN.
7619 ZK=0:ZJ=0:ZA=0:ZL=2^ZM
7620 IFZL<=2THENRETURN
7622 FORZI=0TOZL/4-1:Z1=ZI+ZI+1+ZA:Z2=Z1-1+ZL/2:ZS=Z(Z1)
7623 Z(Z1)=Z(Z2):Z(Z2)=ZS:NEXT
7624 ZL(ZJ,ZK)=ZL:ZA(ZJ,ZK)=ZA:ZJ=ZJ+1:ZL=ZL/2:GOSUB7620
7625 ZJ=ZJ-1:ZL=ZL(ZJ,ZK):ZA=ZA(ZJ,ZK)
7626 ZK=ZK+1:ZL=ZL/2:ZA=ZA+ZL:GOSUB7620:ZK=ZK-1:ZL=ZL(ZJ,ZK):ZA=ZA(ZJ,ZK)
7628 RETURN
READY.
```

**Bild 1. Das Programm „Umordnen"**

Wenn die Schachtelung in die Tiefe ihr Ende gefunden hat, müssen die Operationen auf die zweite Folge in der jeweils darüber liegenden Ebene angewendet werden.

All dies wird mit den folgenden Anweisungen erreicht:

```
100 ZA = 0
200 ZL = 2 ↑ ZM
300 IF ZL < = 2 THEN RETURN
400 REM Vertauschungsfolge
...
500 ZL = ZL/2
510 GOSUB 300
520 ZA = ZA + ZL
530 GOSUB 300
600 RETURN
```

Man erkennt, daß der mit der Anweisung 300 beginnende Programmteil zweimal sich selbst aufruft, entsprechend der durch Halbierung entstehenden Zerlegung in zwei neue Teilfolgen.

### Für jeden Aufruf Zwischenwerte

In einer höheren Programmiersprache, wie z. B. in Pascal, wäre damit die Programmierung im wesentlichen abgeschlossen. In Basic müssen wir jedoch beachten, daß einige der vor dem Selbstaufruf geltenden Variableninhalte, etwa der Inhalt von ZA, während des erneuten Programmdurchlaufes auf einer tieferen Ebene verändert werden, diese Inhalte aber bei Rückkehr auf die höhere Ebene noch bekannt sein müssen. Die später noch benötigten Variablenwerte werden deshalb in unserem Programm vor dem Selbstaufruf in einer Feldvariablen verwahrt, deren Index die Programmebene kennzeichnet. Da in unserem Verfahren je Ebene zweimal ein Selbstaufruf vorkommt, müssen wir sogar eine zweifach indizierte Feldvariable nehmen. So wird der Anfangsindex ZA der jeweiligen Ebene in ZA(ZJ, ZK) gespeichert. ZJ kennzeichnet die Ebene, ZK kennzeichnet, ob der im Feld gespeicherte Wert für die obere oder für die untere Teilfolge in dieser Ebene gelten soll. ZK nimmt dementsprechend nur Werte 0 (oben) oder 1 (unten) an. Das gesamte Programm finden Sie in *Bild 1*. Das Programm läßt erkennen, wie vor und nach jedem Selbstaufruf die für die jeweilige Ebene benötigten Werte von ZA und ZL verwahrt bzw. wieder hervorgeholt werden. Es läßt auch erkennen, wie der Ebenenindex ZJ und der Hälftenindex ZK herauf- und herabgesetzt werden.

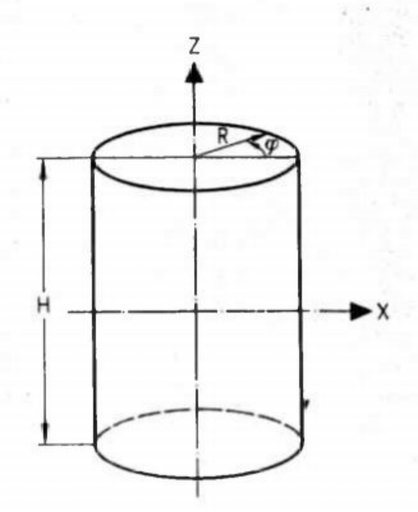



**Bild 2. Mehrfache Integration ist zum Beispiel bei der Berechnung des Trägheitsmomentes eines Kreiszylinders vonnöten**

Das gesamte Programm ist erstaunlich kurz, für den mit rekursiver Technik nicht vertrauten Leser möglicherweise nicht so einfach zu durchschauen. Versuchen Sie es einmal ohne Rekursion. Bei der rekursiven Programmierung in Basic und den damit verbundenen geschachtelten Anweisungen GOSUB ist noch zu beachten, daß Mikrocomputer in der Regel nur eine beschränkte Anzahl von nicht abgeschlossenen GOSUB-Anweisungen gleichzeitig verwalten können. Hinzu kommt, daß der für die GOSUB-Verwaltung zuständige Speicherbereich meist auch für die Verwaltung anderer Anweisungen verwendet wird, zum Beispiel für die Verwaltung der Wiederholungsanweisung FOR... NEXT. Wenn bei rekursiver Programmierung eine größere Anzahl von Ebenen zu erwarten ist, sollte man in solchen Fällen darauf achten, daß möglichst alle Schleifen abgeschlossen sind.

### Rekursion nützlich angewandt

Naheliegend ist die rekursive Programmierung in der Praxis für die Berechnung von Mehrfachintegralen. Als Testbeispiel eignet sich die Berechnung des Trägheitsmomentes eines Kreiszylinders mit der Masse M in bezug auf eine Achse x senkrecht zur Symmetrieachse z (siehe *Bild 2*), das gemäß

$$T_x = M \cdot 2 \cdot \int_0^{H/2} \int_0^{2\pi} \int_0^{R} \left((r \sin \varphi)^2 + z^2\right) r \, dr \, d\varphi \, dz$$

gegeben ist. Das Resultat ist bekannt,

$$T_x = M \cdot \frac{1}{4} \left(R^2 + \frac{1}{3} H^2\right)$$

die numerische Berechnung läßt sich also an diesem Beispiel testen.
Haben wir ein Quadratur-Programm, also ein Programm zur Berechnung bestimmter Integrale zu Hand und wenden wir es für die Berechnung des äußeren Integrals

$$\int_0^{H/2} F(z) \, dz$$

an, dann müssen wir in diesem Programm wiederholt Werte des Integranden

$$F(z) = \int_0^{2\pi} G(z,\varphi) \, d\varphi$$

berechnen. Diese Werte verlangen ihrerseits die Berechnung eines Integrals. Die Berechnung ist mit dem gleichen Quadraturprogramm zu bewältigen. Schließlich verlangt die Berechnung des Integranden für das zweite Integral wiederum eine Quadratur, nämlich die Berechnung des bestimmten Integrals

## Spruch des Monats

„Zwei Dinge machen mir Sorgen: Die Entwicklung von Mikroprozessor-Platinen und Peripherie-Platinen ist durch das BMFT derart gefördert worden, daß der jetzt schon dichte Markt auf diesem Sektor mit einem Überangebot rechnen muß. Zum zweiten ist die jetzige Euphorie nicht angebracht, wenn es nicht gelingt, die neu entwickelten Produkte auch in den Markt zu bringen. Gerade hier treten aber auch die höheren Kosten und häufig auch die größeren Probleme gegenüber der Entwicklung auf. Ich wünschte mir, viele gute Entwickler wären ebenso gute Marktanalytiker."

*H. D. Kref, als Vorstandsmitglied der Interessengemeinschaft Elektronik e. V. zuständig für die BMFT-Förderung, auf einer Mitglieder-Versammlung der I.G.E.L.*

$$G(z,\varphi) = \int_0^R ((r \sin \varphi)^2 + z^2)\, r\, dr$$

Insgesamt ist also ein dreifach geschachtelter rekursiver Aufruf erforderlich. Wir legen für die rekursive Berechnung des Mehrfachintegrals ein Quadraturprogramm zugrunde, das die Gaußsche Quadraturformel verwendet. Soll damit ein Integral

$$\int_a^b f(x)\, dx$$

berechnet werden, so zerlegt man das Intervall [a, b] in gleich lange Teilintervalle. Über jedem dieser Teilintervalle mit Anfangspunkt $x_A$ und Länge 2h wird gemäß

$$\int_{x_A}^{x_A + 2h} f(x)\, dx = h \left\{ f\left(x_A + h - \frac{h}{\sqrt{3}}\right) + f\left(x_A + h + \frac{h}{\sqrt{3}}\right) \right\}$$

der zugehörige Beitrag zum gesamten Integral berechnet. Die nachfolgende Programmversion mit Namen QUADRATUR, die diesen Algorithmus verwendet, ist im Hinblick auf die rekursive Verwendung etwas umständlich geschrieben worden. Anfangswert a und Endwert b müssen vor Aufruf von QUADRATUR in ZA und ZB vorgegeben sein, die Anzahl der Teilintervalle in ZD. Der Integrand z = f(x) wird im Unterprogramm INTEGRAND berechnet. Bei Aufruf von INTEGRAND liegt der Wert x in der Variablen ZX, das Ergebnis z = f(x) soll in die Variable Z gespeichert werden. Das Programm INTEGRAND muß

```
 100 REM QUADRATUR
 101 REM EINGABE: ZA,ZB,ZD  AUSGABE:ZC
 102 REM BERECHNUNG DES BESTIMMTEN INTEGRALS EINER FUNKTION Z=F(ZX) UEBER DEM
 103 REM INTERVALL [ZA,ZB], DAS IN ZD GLEICHABSTANDIGE TEILE ZERLEGT WIRD.
 104 REM WERTE DER FUNKTION WERDEN IM UNTERPROGRAMM ZEILE 140 BERECHNET. DAS
 105 REM NOCH ERGAENZT WWERDEN MUSS. DAS ERGEBNIS WIRD IN ZC ABGESPEICHERT.
 109 REM ANFANGSWERTE
 110 Z3=1/SQR(3):ZH=(ZB-ZA)/ZD/2:ZB=ZA+ZH*(1+Z3):ZA=ZA+ZH*(1-Z3)
 120 ZC=0:ZI=0
 121 REM ANFANG SCHLEIFE
 122 ZJ=0:ZX=ZA
 124 GOSUB142:REM INTEGRAND BERECHNEN
 126 ZC=ZC+Z:IFZJ=1THENGOTO130
 128 ZJ=1:ZX=ZB:GOTO124
 130 ZI=ZI+1:IF ZI>ZD-1THENGOTO134
 132 ZA=ZA+ZH+ZH:ZB=ZB=ZH+ZH:GOTO122
 133 REM ENDE SCHLEIFE
 134 ZC=ZC*ZH:RETURN
 140 REN INTEGRAND
 141 REM BERECHNUNG DES INTEGRANDEN AN DER STELLE ZX UND ABSPEICHERN IN Z
 142 Z=
 149 RETURN
READY.
```

**Bild 3. Quadratur nennt man die Berechnung des Wertes bestimmter Integrale, denn diese Werte können als Flächeninhalt, als Quadrat gedeutet werden**

```
READY.

 2500 REM MULTIQUADGAUS
 2501 REM EIN:ZL,Z1(I),Z2(I),ZD(I) I=[1,ZL]  AUS:ZC  INTERNE VARIABLE:ZA,ZB,ZD,
 2502 REM ZH,ZI,ZJ,ZW,Z3,ZA(I),ZB(I),ZC(I),ZH(I),ZI(I),ZJ(I),ZX(I) I=[1,ZL]
 2503 REM UNTERPROGRAMM ZUR BERECHNUNG EINES MEHRFACHINTEGRALS MIT KONSTANTEN
 2504 REM GRENZEN NACH DEM GAUSS'SCHEN QUADRATURVERFAHREN MIT REKURSION.
 2505 REM DIE VIELFACHHEIT IST IN ZL VORZUGEBEN, DIE UNTEREN GRENZEN IN Z1(I),
 2506 REM DIE OBEREN GRENZEN IN Z2(I), DIE ZERLEGUNGEN IN ZD(I). DIE INTEGRATI-
 2507 REM ONSVARIABLEN WERDEN VON INNEN NACH AUSSEN GEZAEHLT, DIE INNERSTE
 2508 REM VARIABLE HAT DEN INDEX ZL=1.
 2509 REM DER INTEGRAND MUSS IM UNTERPROGRAMM 2550 ALS FUNKTION VON ZX(I)
 2510 REM I=[1,ZL] BERECHNET UND UND IN ZC ABGELEGT WERDEN. VOR AUFFRUF DAS
 2511 REM UNTERPROGRAMM 2550 ERGAENZEN! DAS ENDERGEBNIS WIRD IN ZC ABGELEGT.
 2517 IFZWGOTO2520
 2518 DIMZA(ZL),ZB(ZL),ZH(ZL),ZI(ZL),ZJ(ZL),ZX(ZL),ZC(ZL):Z3=1/SQR(3):ZW=1
 2519 REM PARAMETER
 2520 ZC(ZL)=0:ZA=Z1(ZL):ZB=Z2(ZL):ZD=ZD(ZL):ZH=(ZB-ZA)/ZD/2
 2522 ZB=ZA+ZH*(1+Z3):ZA=ZA+ZH*(1-Z3)
 2523 REM SCHLEIFENANFANG
 2524 ZI=0
 2526 ZJ=0:ZX(ZL)=ZA
 2528 IFZL<=1THENGOSUB2550:GOTO2536
 2529 REM REKURSION
 2530 ZA(ZL)=ZA:ZB(ZL)=ZB:ZH(ZL)=ZH:ZI(ZL)=ZI:ZJ(ZL)=ZJ:ZL=ZL-1
 2532 GOSUB2520
 2534 ZL=ZL+1:ZA=ZA(ZL):ZB=ZB(ZL):ZH=ZH(ZL):ZI=ZI(ZL):ZJ=ZJ(ZL):ZD=ZD(ZL)
 2535 REM ENDE REKURSION
 2536 ZC(ZL)=ZC(ZL)+ZC:IFZJGOTO2540
 2538 ZJ=1:ZX(ZL)=ZB:GOTO2528
 2540 ZI=ZI+1:IFZI>ZD-1GOTO2544
 2542 ZA=ZA+ZH+ZH:ZB=ZB+ZH+ZH:GOTO2526
 2543 REM SCHLEIFENENDE
 2544 ZC=ZC(ZL)*ZH:RETURN
 2550 REM INTEGRAND ALS FUNKTION VON ZX(1),ZX(2),...,ZX(ZL)
 2552 ZC=
 2553 RETURN
READY.
```

**Bild 4. Multiquadgaus rechnet Mehrfachintegrale aus**

entsprechend dem jeweiligen Integranden f(x) noch ergänzt werden. Das Endresultat, das Ergebnis der Quadratur, wird in ZC abgelegt. Soweit die Beschreibung des für die Mehrfachintegration zugrunde liegenden Quadraturprogrammes (*Bild 3*).

Bei der rekursiven Verwendung von QUADRATUR müssen wir, wie bereits am ersten rekursiven Programmbeispiel im einzelnen erläutert, vor und nach dem Selbstaufruf die den einzelnen Ebenen zugehörigen Variablenwerte noch verwahren und wieder hervorholen und die Programmebenen durch eine Indexvariable kennzeichnen. Wie dies im einzelnen erreicht werden kann, zeigt das Programm MULTIQUADGAUS. Durch Vergleich mit QUADRATUR kann der Leser die zusätzlichen Anweisungen und Variablen erkennen (*Bild 4*).

Man beachte dabei, daß jetzt drei unabhängige Variable, nämlich z, φ und r vorkommen, so daß statt einer (Rechner-)Variablen ZX ein Feld ZX(I) eingeführt wurde. Man beachte ferner, daß MULTIQUADGAUS nicht speziell für Dreifachintegrale gedacht ist, sondern für Mehrfachintegrale beliebiger Vielfachheit.
Die Vielfachheit muß beim Aufruf in der Variablen ZL vorgegeben werden. In unserem Testbeispiel, nämlich bei der Berechnung eines Dreifachintegrals, gilt ZL = 3. Das nachfolgend angegebene Hauptprogramm TESTMULTIQUADGAUS zeigt, wie mit dem Unterprogramm MULTIQUADGAUS die Berechnung des Trägheitsmomentes des Kreiszylinders durchgeführt wird. Als Ergebnis ist der genaue Wert und der numerisch berechnete Wert mit der zugehörigen Zerlegung angegeben worden. Bei höheren Vielfachheiten hat man zu bedenken, daß die Rechenzeit mindestens mit der dritten Potenz der Zerlegung anwächst. Als letztes Beispiel für rekursive Programmierung bringen wir in *Bild 7* eine Basic-Fassung des rekursiven Sortierprogrammes QUICKSORT aus dem Buch „Algorithmus + Data Structures = Programs" von N. Wirth.

```
READY.

 10 REM TESTMULTIQUADGAUS
 11 REM BERECHNUNG DES TRAEGHEITSMOMENTES EINES KREISZYLINDERS
 20 ZL=3:FORI=1TO3:READZ1(I),Z2(I):NEXT:REM EINLESEN DER GRENZEN
 30 DATA 0, 1, 0, 6.28318531, 0, 0.5
 40 INPUT"ZERLEGUNGEN";ZD(1),ZD(2),ZD(3)
 41 PRINT:PRINT "DATEN"
 42 PRINT "UNTERE GRENZEN:":PRINTZ1(1),Z1(2),Z1(3)
 43 PRINT "OBERE GRENZEN:":PRINTZ2(1),Z2(2),Z2(3)
 44 PRINT "ZERLEGUNGEN:":PRINTZD(1),ZD(2),ZD(3)
 50 GOSUB2500
 60 PRINT:PRINT"ERGEBNIS"
 61 PRINT"NUMERISCH              EXAKT"
 62 PRINT2*ZC,(1+1/3)*π/4
 70 END

READY.

 2550 REM INTEGRAND ALS FUNKTION VON ZX(1),ZX(2),...,ZX(ZL)
 2552 Z=ZX(3):Y=ZX(2):X=ZX(1):ZC=X*SIN(Y):ZC=ZC*ZC+Z*Z:ZC=ZC*X:RETURN
 2553 RETURN
READY.
```

**Bild 5. Das ist ein Testprogramm für Multiquadgaus**

```
DATEN
UNTERE GRENZEN:
 0              0              0
OBERE GRENZEN:
 1              6.28318531            .5
ZERLEGUNGEN:
 3              3              3

ERGEBNIS
NUMERISCH                  EXAKT
 1.04719755                 1.04719755

READY.
```

**Bild 6. Kein schlechtes Ergebnis – bei genügender Rechenzeit**

```
7100 REM SCHACHTELSORT1
7101 REM EINGABE ZA,ZE,Z(I) I=AZA,ZEÜ  AUSGABE Z(I)  INTERNE VARIABLE ZI,ZJ,
7102 REM ZK,ZS,ZW,Z,ZY,ZE(I),ZI(I) I=A1,26Ü
7103 REM UNTERPROGRAMM ZUM ORDNEN EINER ZAHLENFOLGE Z(I). IN DER GEORDNETEN
7104 REM FOLGE IST JEDES GLIED GROESSER ALS DAS VORHERGEHENDE ODER IHM GLEICH.
7105 REM METHODE: REKURSIVE SCHACHTELUNG. DIE FOLGE WIRD UMGEORDNET DERART,
7106 REM DASS IN EINER ERSTEN TEILFOLGE ALLE GLIEDER ENTHALTEN SIND,
7107 REM DIE KLEINER SIND ALS EIN VERGLEICHSWERT, UND IN EINER ZWEITEN TEIL-
7108 REM FOLGE ALLE GROESSEREN GLIEDER. ALS VERGLEICHSWERT WIRD DAS IN DER
7109 REM MITTE DER FOLGE LIEGENDE GLIEDGEWAEHLT. AUF DIE BEIDEN TEILFOLGEN
7110 REM WIRD DAS VERFAHREN ERNEUT ANGEWANDT. DIE ZERLEGUNG IN ZWEI TEILFOLGEN
7111 REM WIRD IN REKURSIVER WEISE SOLANGE WIEDERHOLT, BIS DIE RESULTIERENDE
7112 REM TEILFOLGE AUS HOECHSTENS ZWEI GLIEDERN BESTEHT.
7113 REM BEI AUFRUF MUSS DER ANFANGSINDEX DER FOLGE IN ZA, DER ENDINDEX IN ZE
7114 REM VORLIEGEN. DAS ORDNEN ERFOLGT INNERHALB DES FELDES Z(I), DAHER NUR
7115 REM WENIG ZUSAETZLICHE SPEICHERBELEGUNG. DIE ZULAESSIGE ANZAHL WIRD DURCH
7116 REM DEN ARBEITSSPEICHER UND DURCH DIE ZULAESSIGE ZAHL ZS VON "GOSUB"-
7117 REM SCHACHTELUNGEN BEGRENZT. DIESEZAHL GGF. IN ANWEISUNG 7120 AENDERN!
7118 IFZA=>ZETHENPRINT"ANFANGSINDEX ZA => ENDINDEX ZE! ERROR":STOP
7119 ZK=0:IFZWTHEN7121
7120 ZS=26:DIMZI(ZS),ZE(ZS):ZW=1
7121 ZI=ZA:ZJ=ZE:Z=Z(INT((ZA+ZE)/2))
7122 IFZ(ZI)<ZTHENZI=ZI+1:GOTO7122
7123 IFZ<Z(ZJ)THENZJ=ZJ-1:GOTO7123
7124 IFZI<=ZJTHENZY=Z(ZI):Z(ZI)=Z(ZJ):Z(ZJ)=ZY:ZI=ZI+1:ZJ=ZJ-1
7125 IFZI<ZJTHEN7122
7126 IFZA>=ZJTHEN7128
7127 ZI(ZK)=ZI:ZE(ZK)=ZE:ZE=ZJ:ZK=ZK+1:GOSUB7121:ZK=ZK-1:ZI=ZI(ZK):ZE=ZE(ZK)
7128 IFZI>=ZETHEN7130
7129 ZA=ZI:GOSUB7121
7130 RETURN
READY.
```

**Bild 7. Quicksort ist ein berühmtes Beispiel rekursiver Programmierung**

rekturtaste zur automatischen oder manuellen Korrektur von Schreibfehlern auf dem Blatt, eine Taste, die die Druckeinheit immer einen Schritt hinter das zuletzt geschriebene Zeichen einer Zeile setzt, und schließlich eine Repeat-Taste zum Wiederholen fast aller Zeichen und Funktionen. Zeilenrückschub, also das Zurückdrehen der Walze, ist im Gegensatz zu Zeilenvorschub nur von Hand möglich.
Die Maschine hat einen eigenen Speicher, der die letzten zwölf eingegebenen Zeichen speichert. Er ermöglicht die automatische Korrektur auf dem Blatt und puffert zu schnell eingegebene Zeichen. Dabei stolperte unsere Testmaschine manchmal über ihre eigenen Beine und läßt ein Zeichen aus. Das kann man aber vermeiden, wenn man die Maschine nur mit etwa 5 Anschlägen pro Sekunde schreiben läßt. Sie braucht dann für eine Schreibmaschinenseite zwischen fünf und zehn Minuten.
Da in der Bedienung zwischen Computer- und Normalbetrieb der Schreibmaschine kaum ein Unterschied besteht, läßt sich das Schreiben mit Computer schnell erlernen. Schreibfehler werden schon auf dem Bildschirm korrigiert, so daß sie gar nicht erst auf dem Blatt erscheinen. Das System erweitert sich mit jedem neu getippten und abgespeicherten Text ganz von selbst. Es übernimmt unbesehen alle auf der Schreibmaschine einmal vorgemachten Tastendrücke und beschriftet nicht nur Briefe und Umschläge, sondern zieht sie auch selbst ein und spuckt sie nach Bearbeitung wieder aus. Der dabei entstehende Lärm entspricht dem Durchschnittslärm jeder anderen normalen Schreibmaschine – leider! Doch muß man ja nicht unbedingt daneben sitzen bleiben.

**Literatur**

Koch, J., *Gandhi*, S.: Anschluß einer Tastatur und Anzeige an ein Mikrocomputersystem. Siemens Technische Mitteilung aus dem Bereich Bauelemente.

# Trace-programm für den TRS-80

Hat man ein Basic-Programm so erstellt, daß es syntaktisch richtig ist und läuft, dann ist in einem letzten Schritt zu prüfen, ob es auch logisch das leistet, was es soll. Als Hilfe stehen dem TRS-80-Anwender leider nur die Befehle STOP, TRON, TROFF zur Verfügung. Dabei ist die Behandlung von Haltepunkten mit dem STOP-Befehl umständlich, und die bloße Verfolgung der durchlaufenen Zeilennummern ist oft zu wenig, um die Wirkung der Befehle zu erkennen.
Das Basic-Programm in *Bild 1* schreibt ein kleines Maschinenprogramm in den Speicherbereich hinter 7FA0 H und baut in 41C4 H eine „Umleitung" des Basic-Interpreters zu diesem Maschinenprogramm auf. Nach dem Lauf dieses Basic-Programms hat es seine Aufgabe getan und wird durch NEW gelöscht. Hat man sich bei der Eingabe nicht vertippt und den Speicherschutz mit MEMORY SIZE =32672 nicht vergessen, so kann man sich nun eine eigene Traceroutine als gewöhnliches Basic-Unterprogramm beginnend mit Zeile 10 schreiben (z. B. wie in *Bild 2*).
Der Basic-Interpreter des TRS-80 springt nach der Beendigung jedes Befehls vor der Abfrage, ob angehalten werden soll, zur Adresse 41C4 H. Dort stand in Level 2 vor dem Start des Basic-Programms aus Bild 1 der Returnbefehl C9 H. Jetzt steht dort die Maschinenbefehlsfolge C3 H, AD H, 7F H, also ein Sprungbefehl nach 7FAD H. Falls sich der Computer im TROFF-Modus befindet, wird dort sofort wieder zurückgesprungen. Im TRON-Modus jedoch ruft das angesprungene Maschinenprogramm die Basicroutine auf, welche ab Zeilennummer 10 beginnt und springt erst nach deren Bearbeitung wieder zurück, um den nächsten Basicbefehl zu bearbeiten. Es versteht sich, daß die Basic-Trace-Routine ab Zeile 10 den Befehl TRON nicht enthalten darf.
Die Beispielroutine aus Bild 2 verwendet die Variablen ST als Flag für Single Step (ST=1) und die Variablen AH und IH als Hilfsvariablen. Sie druckt den Wert der Variablen falls er sich seit dem letzten Basicbefehl geändert hat. Nach einem Anhalten kann durch Drücken einer Taste im Programm fortgefahren werden. *Dipl.-Math. Werner Linsler*

```
100 INPUT "WURDE SPEICHER GESCHUETZT AB 32672?(J/N)";A$
110 IF A$<>"J" THEN STOP
120 REM ZEIGERVERBIEGUNG IN 41C4 H VORBEREITEN
130 POKE 16836,201:POKE 16837,173:POKE 16838,127
140 RESTORE:N=32678:Z=0:S(1)=2523:S(2)=2157:S(3)=2111:S(4)=2530
150 FOR J=250 TO 280 STEP 10
160 SU=0:Z=Z+1
170 FOR I=1 TO 21
180 READ A:SU=SU+A:POKE N,A:N=N+1
190 NEXT I
200 IF SU<>S(Z) THEN PRINT"DATENFEHLER IN ZEILE";J:STOP
210 NEXT J
220 POKE 16836,195:REM ZEIGER VERBIEGEN
230 END
250 DATA 0,145,49,48,58,150,58,34,160,127,235,34,162,
 127,225,209,213,229,213,22,25
260 DATA 30,198,25,209,223,40,11,42,162,127,235,42,160,
 127,175,201,0,0,58,27,65
270 DATA 183,40,239,58,166,127,183,32,24,50,27,65,47,58,
 166,127,42,160,127,34,164
280 DATA 127,225,225,33,57,0,25,229,33,166,127,201,175,
 50,166,127,42,164,127,24,207
290 REM HEXCODES ALS VERGLEICH009131303A963A22A07FEB22A27FE1
 D1D5E5D516191EC619D1DF280B2AA27FEB2AA07FAFC900003A1B41
300 REM B728EF3AA67FB72018321B412F32A67F2AA07F22A47FE1E1213900
 19E521A67FC9AF32A67F2AA47F18CF
```

**Bild 1. Initialisierungsprogramm für die Trace-Routine. Nach dem Laden von der Kassette oder Floppy und dem Start mit RUN kann es mit NEW wieder gelöscht werden, da es eine Maschinenroutine erzeugt. Die REM-Zeilen müssen nicht eingegeben werden**

```
10 PRINT:IF A=AH AND I=IH THEN 60
20 IF A<>AH THEN PRINT"A:";AH;"-";A:AH=A
30 IF I<>IH THEN PRINT"I:";IH;"-";I:IH=I
40 XX$=INKEY$:IF XX$="" THEN 40
50 RETURN
60 IF ST=1 THEN 40 ELSE 50
```

**Bild 2. Beispielprogramm zum Ausdruck sich verändernder Variablen**

Ortwin G. van Eerd

# Programm zur graphischen Darstellung von Daten

Ein Computer erleichtert dem Benutzer die Bewältigung der auf ihn zukommenden Informationsflut. Daten werden eingegeben, gespeichert, verarbeitet und in aufbereiteter Form wieder ausgegeben. Dabei sind die Nachteile der Darstellung größerer Datenmengen in Tabellen und Listen offensichtlich: der Zusammenhang geht oft verloren und die Darstellung wird leicht unübersichtlich. Um hier Abhilfe zu schaffen, kann man Daten auch graphisch darstellen.

Die Darstellung mit Hilfe einer Kurve bietet eine übersichtliche und zusammenhängende Möglichkeit der Abbildung größerer Datenmengen. Im Folgenden ist ein Unterprogramm beschrieben, das beliebige Daten in einer Kurve abbildet. Das Programm läuft auf einem TRS-80 und nutzt insbesondere die graphischen Möglichkeiten (SET(X,Y)) des Gerätes aus. Der Benutzer stellt lediglich die Daten, die dargestellt werden sollen, in einer Matrix zur Verfügung (X- und Y-Werte). Für X und Y können beliebige Werte angegeben werden, die X-Werte müssen jedoch linear aufsteigend sein. Das Programm ermittelt die Skalenwerte für die X- und die Y-Achse und stellt sie am Bildschirm dar. Anschließend werden X- und Y-Werte dem Maßstab des Bildschirms entsprechend ausgegeben (128×48 Bildpunkte). Die Menge der Daten ist beliebig.

```
10 DIM Z(100,1)
20 FOR I=-3.14 TO 6.26 STEP .1
30 Z((I+3.14)*10,0)=I
40 Z((I+3.14)*10,1)=SIN(I)
50 NEXT
60 GOSUB 60000
```

**Bild 1: Dieses Programm berechnet eine Sinusfunktion und legt die errechneten Werte in Z (i, j) ab**

*Bild 1* zeigt als Beispiel eines Hauptprogramms die Berechnung und Darstellung einer Sinusfunktion im Bereich $-\pi$ und $+2\pi$.

### Füllen der Matrix

Der Benutzer stellt in der Matrix Z (i, j) beliebig viele Werte zur Verfügung. In Z (i, 0) die X-Werte, in Z (i, 1) die Y-Werte. Die Dimensionierung muß im Hauptprogramm erfolgen. Dabei ist zu beachten, daß die Matrix um 2 Felder größer sein muß, als die benötigte Anzahl Felder, um das Ende der Matrix erkennen zu können. Es gilt als erreicht, wenn zwei aufeinanderfolgende X-Werte Z (i, 0) und Z (i+1,0) gleich Null sind.
Beispiel: die X-Werte 0,2,4,6,8 und zwei Endekennzeichen ergeben eine Matrix, die mit DIM Z (6,1) oder beliebig größer dimensioniert werden muß. Das Programm führt dann die im Ablaufdiagramm (*Bild 2*) dargestellten Schritte durch. Nach der Ausführung des Programms (*Bild 3*) stehen die Daten nicht mehr in ihrer ursprünglichen Form in Z (i, j) zur Verfügung. Werden sie weiterhin benötigt, müssen sie vorher gesichert werden.
Natürlich kann mit diesem Programm auch mehr als nur eine Kurve ausgegeben werden. Die Programmerweiterungen bzw. -änderungen sind einfach. Für jede Kurve, die zusätzlich ausgegeben werden soll, muß die Dimensionierung von Z (i, j) für die Y-Werte um 1 erhöht werden.
1. Kurve DIM Z (i, 1)
2. Kurve DIM Z (i, 2)...
Der Benutzer muß nun in Z (i, 0) die X-Werte zur Verfügung stellen, in Z (i, 1) die Y-Werte des ersten Polygonzugs, in Z (i, 2) die Y-Werte des zweiten, usw. Folgende Anweisungen sind zusätzlich nötig:

```
60021 IF Z(I, 2) >K THEN K=Z (I, 2)
60031 IF Z (I, 2) <L THEN L=Z (I, 2)
```

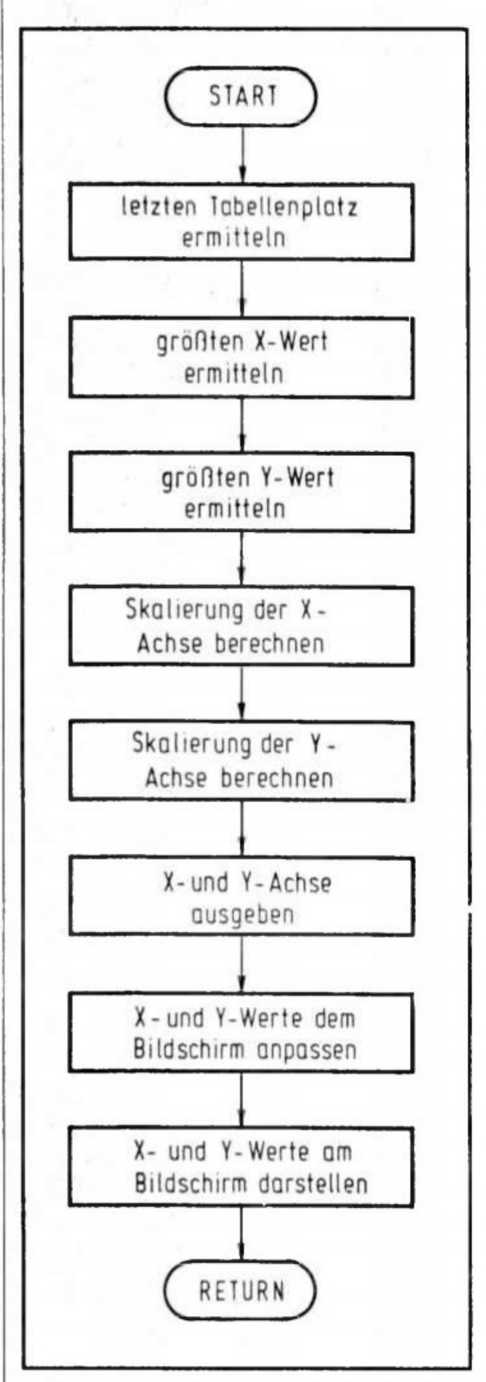



**Bild 2: Das Flußdiagramm zeigt die einzelnen Programmschritte nach Füllen der Matrix**

Die Anweisung in Zeile 60410 muß erweitert werden:
...: Z (I%, 2) = Z (I%, 2)+ABS (L):...
Ebenso die Zeile 60510:
...: Y=40− (Z(I%, 2) −L)/(K−L)*40:SET (X, Y):...

### Programmbeschreibung

60 000
Festlegung von Anfangswerten für I, K, L. Füllen der Ausgabemasken Z1$, Z2$. Dimensionierung der Y-Achse.
60 010
Erkennen, wie weit die Matrix mit Daten gefüllt ist. Ende, wenn zwei aufeinanderfolgende X-Werte gleich Null sind. J erhält den Index des letzten gefüllten Matrixelements.
60 020 – 60 030
Feststellen des größten (K) und des kleinsten (L) Y-Wertes in der Matrix.
60 040
Da das Ende der Matrix noch nicht erreicht ist, wird der Index I umd 1 erhöht.
60 100 – 60 110
Ermittlung der Skala für die X-Achse. 7 Skalenwerte werden ermittelt und in X(0) bis X(6) gespeichert.
60 200 – 60 210
Ermittlung der Skala für die Y-Achse. 14 Skalenwerte werden ermittelt und in Y(0) bis Y(13) gespeichert.
60 300 – 60330
Ausgabe der Bildschirmmaske, d. h. X- und Y-Achse und Punktraster.
60 400 – 60 410
Wenn negative X- oder Y-Werte vorkommen, muß dafür gesorgt werden, daß alle Werte größer oder gleich Null sind.
60 500 – 60 510
Ausgabe des Polygonzug. Enthält die Matrix weniger als 100 Werte, können alle Werte ausgegeben werden (Schrittweite S=1). Bei mehr als 100 Werten, kann nur eine Teilmenge zur Darstellung des Polygonzugs herangezogen werden (S=J/100).
65 000
Rücksprung ins Hauptprogramm.

```
59990  REM ........... U N T E R P R O G R A M M ............
60000  DIMY(13):Z1$="######.##":Z2$=" . . . . . . . . . . . . . . . . . .
       . . . . . . . . . . .":I=0:K=Z(I,1):L=Z(I,1)
60010  IFZ(I,0)=0ANDZ(I+1,0)=0THENJ=I-1:GOTO60100
60020  IFZ(I,1)>KTHENK=Z(I,1)
60030  IFZ(I,1)<LTHENL=Z(I,1)
60040  I=I+1:GOTO60010
60100  XZ=(Z(J,0)-Z(0,0))/6:X(0)=Z(0,0):X(6)=Z(J,0)
60110  FORI%=1TO5:X(I%)=X(I%-1)+XZ:NEXT
60200  YZ=(K-L)/13:Y(0)=L:Y(13)=K
60210  FORI%=1TO12:Y(I%)=Y(I%-1)+YZ:NEXT
60300  CLS:FORI%=0TO13:PRINT@64*I%,"";:PRINTUSINGZ1$;Y(13-I%);:PRINTZ2$;:N
       EXT
60310  FORI%=0TO6:PRINT@965+8*I%,"";:PRINTUSINGZ1$;X(I%);:NEXT
60320  FORI%=0TO42:SET(19,I%):NEXT
60330  FORI%=19TO125:SET(I%,43):NEXT
60400  IFZ(0,0)=>0THEN60410ELSEFORI%=JTO0STEP-1:Z(I%,0)=Z(I%,0)+ABS(Z(0,0)
       ):NEXT
60410  IFL=>0THEN60500ELSEFORI%=0TOJ:Z(I%,1)=Z(I%,1)+ABS(L):NEXT:K=K+ABS(L
       ):L=0
60500  IFJ<=100THENS=1ELSES=J/100
60510  FORI%=0TOJSTEPS:X=(Z(I%,0)-Z(0,0))/(Z(J,0)-Z(0,0))*100+21:Y=40-(Z(I
       %,1)-L)/(K-L)*40:SET(X,Y):NEXT
65000  IFINKEY$=""THEN65000ELSERETURN
```

**Bild 3: Das Unterprogramm zur Darstellung der errechneten Werte auf dem Bildschirm**

**Tabelle: Die Variablenliste des Programms**

| Variable | Bedeutung |
|---|---|
| J | letzter Tabellenplatz |
| K | größter Y-Wert |
| L | größter Y-Wert |
| XZ | Faktor zur Ermittlung der Skalenwerte des X-Achse |
| X (0) – X (5) | Skalenwerte der X-Achse |
| YZ | Faktor zur Ermittlung der Skalenwerte der Y-Achse |
| Y (0) – Y (13) | Skalenwerte der Y-Achse |
| X | Bildpunkt-Koordinate |
| Y | Bildpunkt-Koordinate |
| Z1$ | Maske für PRINT USING der Skalenwerte |
| Z2$ | Punktraster für Ausgabebild |
| S | Schrittweite zur Ausgabe der Matrix |

# POP mit Flags

Sehr häufig tritt bei 8080-, 8085- oder Z80-Software das Problem auf, daß man sich den Akku erhalten möchte, die Flags jedoch für nachfolgende Verzweigungen in einem veränderten Zustand belassen möchte. Die PUSH-POP-Kombination führt hier nicht zum gewünschten Ergebnis, da die Flags dabei ja auch wieder in den alten (unerwünschten) Zustand zurückgesetzt werden. Mit nur zwei zusätzlichen Befehlen läßt sich der Akku wiederherstellen, ohne die Flags zu zerstören. Anstelle von POP AF (bzw. POP PSW) tritt die Befehlsfolge EX (SP).HL/LD A,H/POP HL (bzw. XTHL/ MOV A,H/POP H).

Das Anwendungsbeispiel im *Bild* zeigt die Abfrage eines Doppelregisters (hier das Registerpaar BC) nach Null. Das Ergebnis dieser Operation setzt das Zero-Flag nur dann, wenn der Inhalt des Registerpaares Null ist. *Herbert Paluch*

```
Z80              8080/8085

PUSH AF          PUSH PSW        ;AKKU RETTEN
LD A,B           MOV A,B         ;HIGHER BYTE LADEN
OR C             ORA C           ;VERGLEICH MIT LOWER BYTE
EX (SP),HL       XTHL            ;AKKU NACH H und HL RETTEN
LD A,H           MOV A,H         ;AKKU RESTAURIEREN
POP HL           POP H           ;HL EBENSO
JP Z,BCNULL      JZ BCNULL       ;JUMP WENN BC GLEICH 0
```

**Kleine Unterschiede zwischen Z80 und 8080/8085 sind beim Zurückholen des Akkus vom Stack ohne Flag-Zerstörung zu sehen**